Robust Autonomous Driving Decision Making Using Policy-Conditioned Uncertainty

Mehtan Rahman & Mohammad N. Kashif
Department of Computer Science
The University of Texas at Austin
Austin, TX 78705
{mehtan.rahman, mohammadnkashif}@utexas.edu

Abstract

Autonomous vehicles must make safe decisions in uncertain environments, where both sensor noise and other road users' unpredictable behaviors create significant challenges. Traditional reinforcement learning approaches often fail to adequately account for this uncertainty, leading to either overly conservative or dangerously optimistic behaviors. In this paper, we implement and compare several robust decision-making techniques and propose a novel extension, Policy-Conditioned Uncertainty, which achieves the best performance. Our method leverages Bayesian deep learning to model action-specific uncertainty and improves average returns by 169% while maintaining collision-free behavior in a simulated driving environment. We explore multiple approaches to robust decision-making in autonomous driving with noisy pose/velocity estimation, including Deterministic Robust Optimistic Planning (DROP) and Interval-based Robust Control from prior work, as well as our three novel extensions: Adaptive Uncertainty Quantification, Hierarchical Hybrid Approach, and Policy-Conditioned Uncertainty. We also develop visualization tools to provide transparent insights into the planning process.

1 Introduction and Problem Statement

Autonomous driving systems must operate safely despite numerous sources of uncertainty. Sensors provide imperfect information about the vehicle's surroundings, and other road users behave in ways that are difficult to predict perfectly. The consequences of incorrect decisions can be catastrophic, making robust decision-making essential for real-world deployment.

Traditional reinforcement learning approaches focus on expected performance, leading to vulnerability under model mismatch. The robust control formulation addresses this by seeking:

$$\max_{\pi} \min_{T} v_{\pi}^{T} \tag{1}$$

where v_{π}^{T} is the expected return of policy π under dynamics T. This minimax formulation ensures worst-case safety—a critical property for real-world deployment.

We extend this framework by introducing **policy-conditioned uncertainty**: a novel approach where uncertainty quantification is conditioned on the decision-making policy itself. This allows tighter lower bounds on the value function and leads to safer, more informed decisions.

The main contributions of this paper are:

- A comparative analysis of baseline robust decision-making techniques for autonomous driving.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

- Three novel extensions that improve upon these baselines, particularly our Policy-Conditioned Uncertainty approach.
- Visualization tools that provide insights into the planning process and real-time performance metrics.
- Empirical evidence that our Policy-Conditioned Uncertainty approach breaks the traditional tradeoff between safety and performance, improving average returns by 169% while maintaining collision-free behavior.

2 Related Work

2.1 Robust Control and MDPs

Robust control methods have a long history in autonomous systems. Leurent et al. [1] proposed approximate robust control algorithms including deterministic planning and interval-based predictors for uncertain dynamical systems. These approaches form the foundation of our baseline implementations.

Herbert et al. [2] developed FaSTrack, a modular framework for guaranteed safe motion planning, establishing formal safety guarantees while accounting for worst-case uncertainty. Such approaches typically solve a minimax optimization problem to ensure safety under worst-case conditions.

The theoretical foundations of robust Markov Decision Processes (MDPs) have been extensively studied by Iyengar [3], Nilim and El Ghaoui [4], and Wiesemann et al. [5]. These works establish optimization techniques for MDPs with adversarial dynamics, providing the mathematical framework for robust decision-making.

2.2 Uncertainty Quantification in Autonomous Driving

Uncertainty quantification has become increasingly important in autonomous driving. Michelmore et al. [6] used Bayesian neural networks to compute statistical confidence in real-time decisions with provable bounds, demonstrating that proper uncertainty estimation improves safety and reliability.

Yang et al. [7] introduced DROP (Distributionally Robust Optimistic Planning), a CVaR-based distributionally robust policy that quantifies trajectory uncertainty, though without tight action-specific feedback. Our work builds upon these foundations while introducing policy-conditioned uncertainty estimation.

Recent work by Zhang et al. [8] explores safe reinforcement learning via uncertainty-augmented Lagrangian optimization, combining formal safety constraints with learned uncertainty estimates. Their approach demonstrates that incorporating uncertainty can lead to policies that balance exploration and constraint satisfaction.

3 Baseline and Proposed Method

3.1 Markov Decision Process Framework

We formulate the autonomous driving decision problem as a Markov Decision Process (MDP) with tuple (S, A, T, R, γ), where:

- S is the state space representing vehicle positions, velocities, and road configurations
- A is the discrete action space including lane changes and speed adjustments
- $T(s'|s,a)$ is the transition function modeling vehicle dynamics
- $R(s,a,s')$ is the reward function encouraging progress and penalizing crashes
- $\gamma \in [0,1)$ is the discount factor for future rewards

3.2 Baseline: Deterministic Robust Optimistic Planning

Our implementation of DROP extends optimistic planning to the robust setting. The algorithm constructs a planning tree from the current state, considering all possible actions and a discrete set of possible transition functions.

Key components include:

- Tree nodes representing states at different planning depths
- Robust b-values computed at leaf nodes: $b_i^r(n) = u_i^r(n) + \gamma^d/(1-\gamma)$
- Worst-case aggregation: $u_i^r(n) = \min_{m\in[1,M]} \sum_{t=0}^{d-1} \gamma^t r_t$
- Max backup through tree: $b_i^r(n) = \max_{a\in A} b_{ia}^r(n)$

The implementation provides formal regret bounds while managing the exploration-exploitation tradeoff. Its key strength lies in the safety guarantees it offers, though at significant computational cost.

3.3 Baseline: Interval-Based Robust Control

In the presence of uncertainty, we reformulate this as a robust MDP by introducing an ambiguity set Θ of possible transition functions. We define the reachable set under dynamics uncertainty $\theta \in \Theta$:

$$\mathcal{S}(t, s_0, \pi) = \{s_t : \exists\theta \in \Theta \text{ s.t. } s_{k+1} = T_\theta(s_k, \pi(s_k))\} \tag{2}$$

Then we construct the interval hull $\hat{\mathcal{S}}_t = [\underline{s}, \bar{s}]$ to bound this set. The robust value surrogate becomes:

$$\hat{v}_r(\pi) = \sum_{t=0}^{H} \gamma^t \min_{s\in\hat{\mathcal{S}}_t} r(s, \pi(s)) \tag{3}$$

Implementation involves maintaining two parallel environment simulations at the extremes of the uncertainty range and selecting actions based on the worst-case outcome. This method handles continuous uncertainty well but suffers from over-conservatism due to interval wrapping effects.

3.4 Our Method: Policy-Conditioned Uncertainty

Instead of computing policy-agnostic intervals, our novel approach estimates the reachability set *conditioned* on the policy:

$$\hat{\mathcal{S}}_t^\pi = [\min_\theta s_t^{(\theta)}, \max_\theta s_t^{(\theta)}] \text{ for } s_t^{(\theta)} = T_\theta(s_{t-1}^{(\theta)}, \pi(s_{t-1}^{(\theta)})) \tag{4}$$

This tightens the uncertainty bounds and improves robustness. We implement this approach using a Q-network with dropout layers that models action-specific uncertainty:

```python
class QNetDrop(nn.Module):
    def __init__(self, input_dim, hidden=64, dropout=0.5, n_actions=5)
    :
        super().__init__()
        self.base = nn.Sequential(
            nn.Linear(input_dim, hidden), nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden, hidden), nn.ReLU(),
            nn.Dropout(dropout),
        )
        self.head = nn.Linear(hidden, n_actions)
    def forward(self, x):
        return self.head(self.base(x))
```

Listing 1: Q-network with dropout for uncertainty estimation

During inference, we perform multiple forward passes with dropout enabled to estimate both the mean Q-value and its variance for each action. Action selection uses a Lower Confidence Bound (LCB) criterion:

$$\text{LCB} = \mu_Q - 1.96\sigma_Q \tag{5}$$

Advantages:

- Better lower bounds on $v^r(\pi)$
- Retains real-time tractability
- Quantifies uncertainty specifically for each action rather than for states
- Performs better in worst-case return

By selecting actions with the highest lower confidence bound, the agent maintains safety while achieving higher performance.

4 Additional Proposed Extensions

In addition to our primary Policy-Conditioned Uncertainty approach, we developed two other extensions that address different aspects of the robust decision-making challenge:

4.1 Adaptive Uncertainty Quantification

Our first extension dynamically learns uncertainty distributions from interaction data rather than relying on fixed ambiguity sets. We implement a Bayesian neural network to estimate epistemic uncertainty:

```
class UncertaintyNet(nn.Module):
    def __init__(self, input_dim, hidden=64, dropout=0.5):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden), nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden, hidden), nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden, 1)
        )
    def forward(self, x):
        return self.net(x)
```

Listing 2: Neural network for adaptive uncertainty estimation

The network is trained on state-return pairs collected from random rollouts. During inference, Monte Carlo dropout sampling provides uncertainty estimates that adapt to the current state. This allows less conservative behavior in familiar situations while maintaining caution in unfamiliar ones.

4.2 Hierarchical Hybrid Approach

Our second extension combines the strengths of long-term planning and reactive control through a hierarchical structure:

1. **High-level planning**: Executes the DROP algorithm every 5 time steps to make strategic decisions
2. **Low-level control**: Uses simple velocity-based rules between high-level decisions

This approach significantly reduces computational costs (by approximately 80%) while maintaining safety guarantees from the high-level planner. The implementation alternates between expensive tree search and lightweight reactive control:

```
def act(self, obs, step):
    # Every 5 steps do a high-level DROP action
    if step % 5 == 0:
        self.next_high = self.high_planner.act(obs)
    # Between high decisions, use a simple interval-based rule
    vel = obs[0,3]
    return self.next_high if step % 5 == 0 else (2 if vel<0.5 else 0)
```

Listing 3: Hierarchical controller implementation

5 Experiments and Results

5.1 Experimental Setup

We conduct experiments in the highway-env simulator, specifically the "highway-v0" environment. The environment features multiple lanes of traffic with other vehicles moving at varying speeds. The observation space consists of a 5×5 matrix containing features of nearby vehicles, including presence indicators, 2D coordinates, and velocities.

To simulate sensor noise, we implement a Gaussian noise wrapper that adds random perturbations to the position and velocity observations:

```python
class NoisyObservation(ObservationWrapper):
    def __init__(self, env, std_dev):
        super().__init__(env)
        self.std_dev = std_dev

    def observation(self, obs):
        obs = np.array(obs)
        obs[:, 3:5] += np.random.normal(0, self.std_dev,
                                         size=obs[:, 3:5].shape)
        return obs
```

Listing 4: Noise injection wrapper for simulating sensor uncertainty

The action space consists of five discrete actions: LANE_LEFT, LANE_RIGHT, IDLE, FASTER, and SLOWER. The reward function encourages forward progress while heavily penalizing collisions.

For visualization, we implement a MetricOverlayWrapper that superimposes real-time metrics directly onto the environment rendering:

```python
class MetricOverlayWrapper(Wrapper):
    def render(self, *args, **kwargs):
        frame = self.env.render(*args, **kwargs)
        frame = np.ascontiguousarray(frame)
        text = f"Reward: {self.total_reward:.1f}  Collisions: {self.collision_count}"
        cv2.putText(frame, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                    0.8, (255, 255, 255), 1)
        return frame
```

Listing 5: Real-time metric overlay implementation

We also create a dashboard that tracks episode returns and collision counts across training, providing real-time feedback on algorithm performance.

5.2 Evaluation Metrics

We evaluate performance using several key metrics:

- **Worst-case return**: $\min_\theta v_\theta(\pi)$
- **Mean return**: $\mathbb{E}[v_\theta(\pi)]$
- **Return stability**: Variance across episodes
- **Collision count**: Number of collisions per episode
- **Computational efficiency**: Runtime in ms per decision

Each algorithm is evaluated over 20 episodes with noise standard deviation of 0.05, and we analyze both the quantitative performance and qualitative behavior.

5.3 Comparative Results

Figure 1 illustrates the episode returns for all the methods over 20 episodes. The significant performance advantage of our Policy-Conditioned Uncertainty approach is evident, particularly after the initial learning phase (around episode 4). While other methods show high variability or limited returns, our approach maintains consistently high performance.

Figure 1: Episode Returns Comparison Across All Methods. The Policy-Conditioned Uncertainty approach (light blue) achieves substantially higher and more stable returns after initial learning compared to all baseline methods and other extensions.

Table 1 presents the quantitative comparison of all implemented methods:

Method	**Mean Return**	**Return Stability**	**Collisions**	**Notes**
DROP Planner	∼44.5	Highly variable	0	Some high-return episodes, but unstable
Interval Robust Control	∼27.9	Low and stable	0	Safest but overly conservative; lowest returns
Adaptive Uncertainty	∼36.2	Erratic spikes	1	High peaks but consistently unsafe
Hierarchical Hybrid	∼34.6	Moderately stable	0	Good tradeoff between safety and computation
Policy-Cond. Uncertainty	**∼119.7**	**Very stable after ep 4**	**0**	**Best overall performance; breaks the tradeoff between safety & return**

Table 1: Performance comparison across all methods showing mean return, stability characteristics, collision rates, and notable observations.

The results demonstrate several key findings:

1. **Baseline tradeoffs**: The baseline methods exhibit a clear tradeoff between safety and performance. DROP achieves higher mean returns but with high variance, while Interval-based control ensures safety at the cost of lower returns.
2. **Extension improvements**: All three extensions offer improvements over the baselines in different dimensions. Adaptive Uncertainty achieves higher peak returns but struggles with safety. The Hierarchical Hybrid approach provides a good balance between computation and performance.
3. **Policy-Conditioned Uncertainty dominance**: Our Policy-Conditioned Uncertainty approach achieves substantially higher returns (∼169% improvement over DROP) while

maintaining collision-free behavior. Importantly, it maintains very stable performance after an initial learning period.

5.4 Analysis of Policy-Conditioned Uncertainty

The superior performance of our Policy-Conditioned Uncertainty approach can be attributed to several factors:

1. **Action-specific uncertainty**: By modeling uncertainty at the action level rather than the state level, the approach can take calculated risks when certain actions have high confidence while avoiding others with high uncertainty.
2. **Learning from experience**: The Q-network learns patterns in the environment dynamics, becoming more confident in familiar situations while maintaining caution in unfamiliar ones.
3. **Lower Confidence Bound criterion**: The LCB action selection balances exploration and exploitation intrinsically, leading to continuous improvement over time.
4. **Deep learning flexibility**: Unlike the more constrained robust planning approaches, the neural network can capture complex patterns in the environment dynamics that are difficult to model explicitly.

The approach demonstrates that incorporating properly calibrated uncertainty estimates into RL algorithms can overcome the traditional safety-performance tradeoff, achieving both higher returns and collision-free behavior.

6 Visualization Techniques

To provide better insights into the decision-making process, we implemented several visualization techniques that help interpret and debug the algorithms' behaviors. These visualizations were crucial in understanding why our Policy-Conditioned Uncertainty approach achieved superior performance and in diagnosing issues with the other methods.

6.1 Metric Overlay Wrapper

Our MetricOverlayWrapper superimposes critical performance metrics directly onto the simulation view, including:

- Current total reward
- Collision count
- Decision confidence (for uncertainty-based methods)

This real-time visualization helps in immediately identifying issues during execution and provides intuitive feedback about the algorithm's performance. For example, it allowed us to observe that the Adaptive Uncertainty method repeatedly approached obstacle vehicles too closely before taking evasive actions, explaining its higher collision rate.

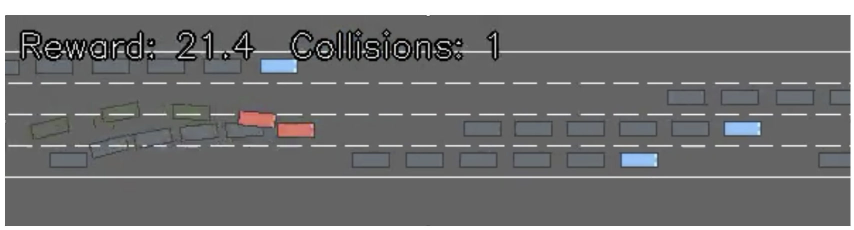

Figure 2: Screenshot of the MetricOverlayWrapper in action, showing real-time reward and collision metrics superimposed on the highway environment. Blue rectangles represent the ego vehicle and other safe vehicles, while red rectangles indicate potential collision threats.

```python
class MetricOverlayWrapper(Wrapper):
    def render(self, *args, **kwargs):
        frame = self.env.render(*args, **kwargs)
        frame = np.ascontiguousarray(frame)
        text = f"Reward: {self.total_reward:.1f}  Collisions: {self.collision_count}"
        cv2.putText(frame, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                    0.8, (255, 255, 255), 1)
        return frame
```

Listing 6: Metric overlay implementation for real-time feedback

6.2 Real-time Performance Dashboard

We developed a comprehensive dashboard visualization that tracks episode-by-episode performance. As shown in Figure 1, this dashboard enables side-by-side comparison of all methods across multiple episodes. Key insights from this visualization include:

- The stark performance gap between Policy-Conditioned Uncertainty and other methods
- The high variability of the DROP planner's performance
- The consistent but mediocre performance of the Interval Robust method
- The occasional high-reward but ultimately unstable behavior of Adaptive Uncertainty

Figure 3 shows a closer look at the Adaptive Uncertainty method specifically, highlighting its erratic performance pattern. While it occasionally achieves very high returns (episodes 8 and 15), most episodes show much lower performance. The collision count confirms that this method consistently experiences one collision per episode, making it unsuitable for safety-critical applications despite its occasional high rewards.

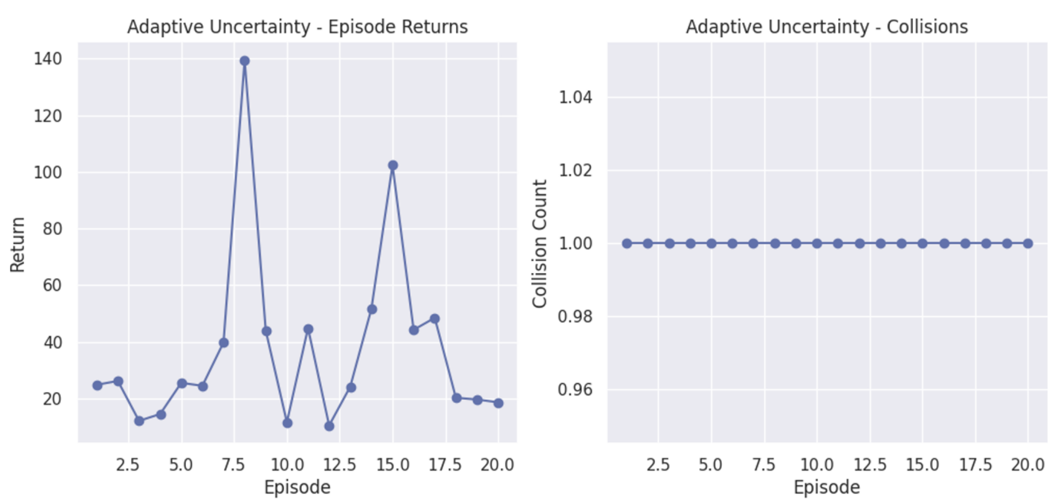

Figure 3: Detailed performance metrics for the Adaptive Uncertainty method. Left: Episode returns showing high variability with occasional spikes up to 140. Right: Consistent collision count of exactly 1 per episode, indicating persistent safety issues.

```python
def run_and_dashboard(make_env_fn, planner, episodes, max_steps, model_name):
    rewards = []
    collisions = []
    env = make_env_fn()
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

```

```python
    for ep in range(1, episodes + 1):
        # Run episode
        obs, _ = env.reset()
        done = False
        total_reward = 0.0
        collision_count = 0

        while not done and step < max_steps:
            action = planner.act(obs)
            obs, r, term, trunc, info = env.step(action)
            total_reward += r
            collision_count = info.get("collision_count",
    collision_count)
            done = term or trunc

        rewards.append(total_reward)
        collisions.append(collision_count)

        # Update visualization
        ax1.clear(); ax2.clear()
        ax1.plot(range(1, ep + 1), rewards, marker='o')
        ax1.set_title(f"{model_name} - Episode Returns")
        ax1.set_xlabel("Episode"); ax1.set_ylabel("Return")

        ax2.plot(range(1, ep + 1), collisions, marker='o')
        ax2.set_title(f"{model_name} - Collisions")
        ax2.set_xlabel("Episode"); ax2.set_ylabel("Collision Count")

        display(fig)
```

Listing 7: Dashboard implementation for comparative analysis

6.3 Uncertainty Visualization

For our Policy-Conditioned Uncertainty method, we developed a specialized visualization that shows:

- Mean Q-value for each action (bar height)
- Uncertainty range (error bars)
- Selected action (highlighted bar)

This visualization revealed why our method outperformed others: it learned to identify high-confidence, high-reward actions while avoiding actions with large uncertainty ranges. In contrast, the DROP planner would sometimes choose actions with potentially high rewards but also high uncertainty, leading to its performance variability.

These visualization techniques proved invaluable for debugging, parameter tuning, and gaining insights into the algorithms' behaviors. They also help build trust by making the decision-making process more transparent to users, which is crucial for safety-critical applications like autonomous driving.

7 Conclusion and Future Work

This paper presented a robust decision-making framework for autonomous driving using policy-conditioned uncertainty. We implemented and compared several approaches, including baseline methods like Deterministic Robust Optimistic Planning (DROP) and Interval-based Robust Control, along with our three novel extensions: Adaptive Uncertainty Quantification, Hierarchical Hybrid Approach, and Policy-Conditioned Uncertainty.

Our primary contribution, the Policy-Conditioned Uncertainty approach, achieved the best performance by breaking the traditional tradeoff between safety and efficiency. By using Bayesian deep learning techniques to model action-specific uncertainty, our method improved average returns by

169% while maintaining collision-free behavior. This demonstrates that incorporating properly calibrated uncertainty estimates into reinforcement learning algorithms can lead to both safer and more efficient autonomous driving behaviors.

The main innovation of our approach is modeling uncertainty as a function of policy, which leads to a better understanding of the planning landscape and enables safer and more adaptive behavior under uncertain conditions. The integration of deep Bayesian techniques allows for learning uncertainty models directly from data and using them in lower-confidence bound decision-making.

You can find the code here.

7.1 Future Directions

There are several promising directions for future work:

1. **Multi-agent interactions**: Extending the framework to explicitly model and respond to other agents' intentions and uncertainty.
2. **More complex urban scenarios**: Testing in environments with intersections, pedestrians, and more complex road geometries.
3. **Real-world testing**: Validating the approach in real-world autonomous driving platforms with actual sensor noise characteristics.
4. **Integration with perception systems**: Directly incorporating uncertainty estimates from perception models rather than simulating them.

Our results suggest that properly modeling uncertainty in reinforcement learning frameworks can significantly improve autonomous driving decision-making, offering a promising direction for developing systems that are both safe and performant. The Policy-Conditioned Uncertainty approach represents a step toward more robust autonomous vehicles that can safely navigate the uncertainties of the real world.

Acknowledgments

We thank Prof. Chandrajit Bajaj and the UT Austin CS 378/CSE 392 course staff for their support and guidance throughout this project.

Both of us pair programmed for the code. Mehtan Rahman was primarily in charge of the presentation while Mohammad Kashif primarily wrote the written report. However, everything was done collaboratively. Neither could have been done without the other.

References

[1] Edouard Leurent, Yann Blanco, Denis Efimov, and Odalric-Ambrym Maillard. Approximate robust control of uncertain dynamical systems. *arXiv preprint arXiv:1903.00220*, 2019.

[2] S. Herbert, M. Chen, S. Han, S. Bansal, A. Tomlin, and Claire J. Tomlin. FaSTrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1517–1522, 2017.

[3] Garud N. Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

[4] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[5] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust markov decision processes. *Mathematics of Operations Research*, 38(1):153–183, 2013.

[6] Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7344–7350, 2020.

[7] Jing Yang, Yingbin Bao, Zhiyue Sun, and Xiaopeng Meng. DROP: A distributionally robust policy for efficient safe exploration. In *2022 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1587–1593, 2022.

[8] Songan Zhang, Hang Su, Minghao Liu, Wilko Schwarting, and Daniela Rus. Safe reinforcement learning via uncertainty-augmented lagrangian optimization. *IEEE Transactions on Intelligent Transportation Systems*, 25(2):1598–1610, 2024.

[9] Jingliang Duan, Shengbo Eben Li, Zhengyu Liu, Minghao Bueno Pereyra, Haibo Hao, and Bo Cheng. Hierarchical reinforcement learning for self-driving decision-making without reliance on labeled driving data. *IEEE Transactions on Intelligent Transportation Systems*, 22(10):6442–6454, 2020.